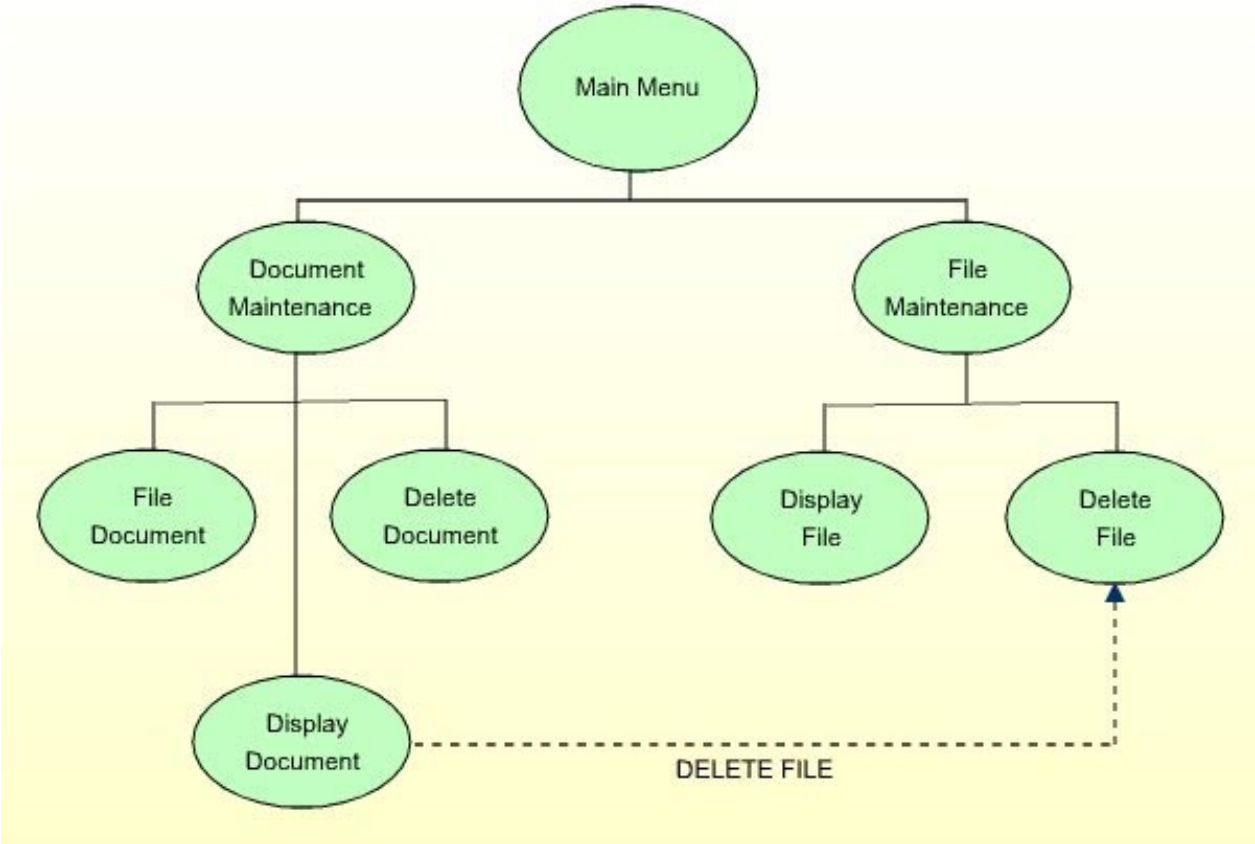
SYSNCP - Introduction

Applications which enable users to move from one activity to another activity by using direct commands far exceed in usability the ones which force the user to navigate through menu hierarchies to a desired activity.



The figure above illustrates the advantage of using direct commands. In an application in which menu hierarchies form the basis for navigation, a user wishing to advance from the Display Document facility to the Delete File facility would have to return to the Main Menu via the document branch and then enter the file branch. This is clearly less efficient than accessing the Delete File facility directly from the Display Document facility.

Below is information on:

- Object-Oriented Data Processing
- Features of the Command Processor
- Components of the Command Processor
- What is a Command?
- Creating a Command Processor

Object-Oriented Data Processing

The Natural command processor is used to define and control navigation within an application. It could be used, for example, to define a command DISPLAY DOCUMENT to provide direct access to the Display Document facility. When a user enters this command string in the command line of a screen (for which this command is allowed), the Natural command processor processes the input and executes the action(s) assigned to the command.

In contrast to menu-driven applications, the command-driven applications implemented with the Natural command processor take a major step toward object-oriented data processing. This approach has the following advantages:

- The design of an application need not depend on the way in which a certain result can be reached, but only on the desired result itself. Thus, the design of an application is no longer influenced by the processing flow within its components.
- The processing units of an application become independent of one another, making application maintenance easier, faster and much more efficient.
- Applications can be easily expanded by adding independent processing units. The resulting applications are, therefore, not only easy to use from an end-user’s view, but also easier to create from a programmer’s view.

The Natural command processor has the following additional benefits:

- **Less Coding**
  Instead of having to repeatedly program lengthy and identically structured statement blocks to handle the processing of commands, you only have to specify a PROCESS COMMAND statement that invokes the command processor; the actual command handling need no longer be specified in the source code. This considerably reduces the amount of coding required.
- **More Efficient Command Handling**
  As the command handling is defined in a standardized way and in one central place, the work involved in creating and maintaining the command-processing part of an application can be done much faster and much more efficiently.
- **Improved Performance**
  The Natural command processor has been designed with particular regard to performance aspects: it enables Natural to process commands as fast as possible and thus contributes to improving the performance of your Natural applications.

Features of the Command Processor

The Natural command processor provides numerous features for efficient and user-friendly command handling:

- **Flexible Handling of Commands**
  You can define aliases (that is, synonyms for keywords), and abbreviations for frequently used commands.

- **Automatic Check for Uniqueness of Abbreviated Keywords**
  The command processor automatically compares every keyword you specify in SYSNCP with all other keywords and determines the minimum number of characters in each keyword required to uniquely identify the keyword. This means that, when entering commands in an application, users can shorten each keyword to the minimum length required by the command processor to distinguish it from other keywords.

- **Local and Global Validity of Commands**
  You can specify in SYSNCP whether the action to be performed in response to a specific command is to be the same under all conditions or situation-dependent. For example, you can make the action dependent on which program was previously issued. In addition, you can define a command to be valid under one condition but invalid under another.

- **Error Handling for Invalid Commands**
  You can attach your own error-handling routines to commands or have error input handled by Natural.

- **Functional Security**
  With Natural Security, library-specific and user-specific conditions of use can be defined for the tables generated with SYSNCP. Thus, for your Natural applications you can allow or disallow specific functions or keywords for a specific user. This is known as functional security.
  See also the section Functional Security in the Natural Security documentation.

- **Help Text**
  In SYSNCP, you can attach help text to a keyword or a command. Then, by specifying a PROCESS COMMAND ACTION TEXT statement, you can return command-specific help text to the program.

- **Online Testing of Command Processing**
  If the execution of a command does not produce the intended result, you can find out why the command was not processed correctly by using the PROCESS COMMAND statement (see the Natural Statements documentation) and the EXAM* sample test programs (source form) provided in the Library SYSNCP. The endings of the EXAM-* program names appear as abbreviations at the top border line of the relevant action windows (for example, EXAM-C appears as **C**).

  To test a command processor at runtime

  1. Enter the direct command EXAM to list all test programs.
     The "Demonstrate PROCESS COMMAND Statement" window is displayed.
  2. Enter Function Code **O** to open a processor.
  3. Enter the name of the processor.
  4. Choose any of the Functions Codes listed (for example, C for CHECK) to apply command actions.
  5. Enter Function Code **Q** to close the processor.

Components of the Command Processor

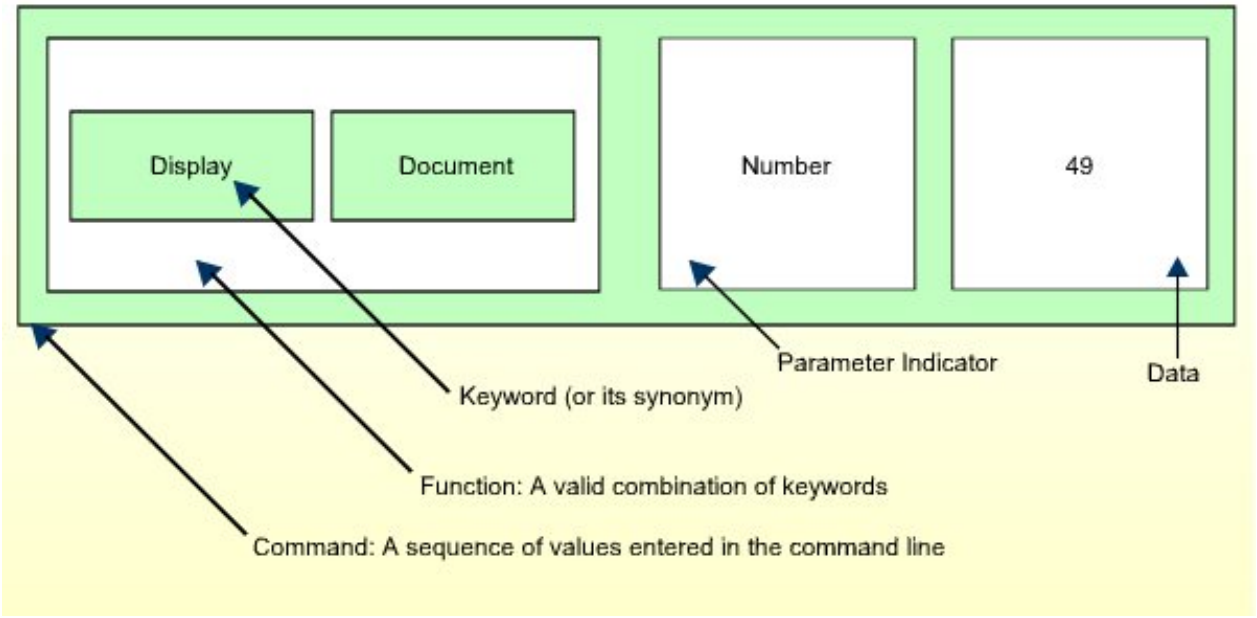
The Natural command processor consists of two parts: a development part and a runtime part:

- The development part is the utility SYSNCP, which is described in this section. With the utility SYSNCP you define commands (as described below) and the actions to be performed in response to the execution of these commands. From your definitions, SYSNCP generates decision tables which determine what happens when a user enters a command. These tables are contained in a Natural member of type Processor.
- The runtime part is the statement PROCESS COMMAND, which is described in the Natural Statements documentation. This statement is used to invoke the command processor within a Natural program. In the statement, you specify the name of the processor to be used to handle the command input by a user at this point.

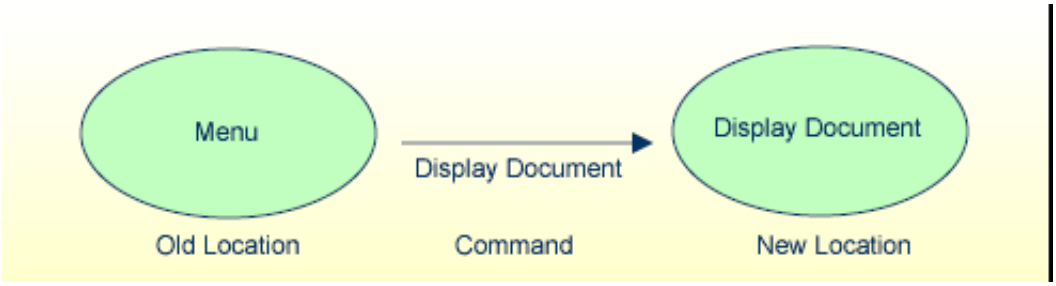
What is a Command?

A command is any sequence of values entered in the command line which is recognized and processed by an application. Commands can contain up to three elements:

- **Function:**
  One or more valid keywords. For example, MENU or DISPLAY DOCUMENT.
- **Parameter Indicator:**
  Optional. A keyword which introduces command data.
- **Command Data:**
  Information to be sent to a function. Command data can be alphanumeric or numeric, for example, the name or the number of the file to be displayed.



Commands are always executed from a situation within an application; the position where this situation is reached is referred to as a location. Commands take the user from one location to another location; thus, each command can be viewed as a vector:

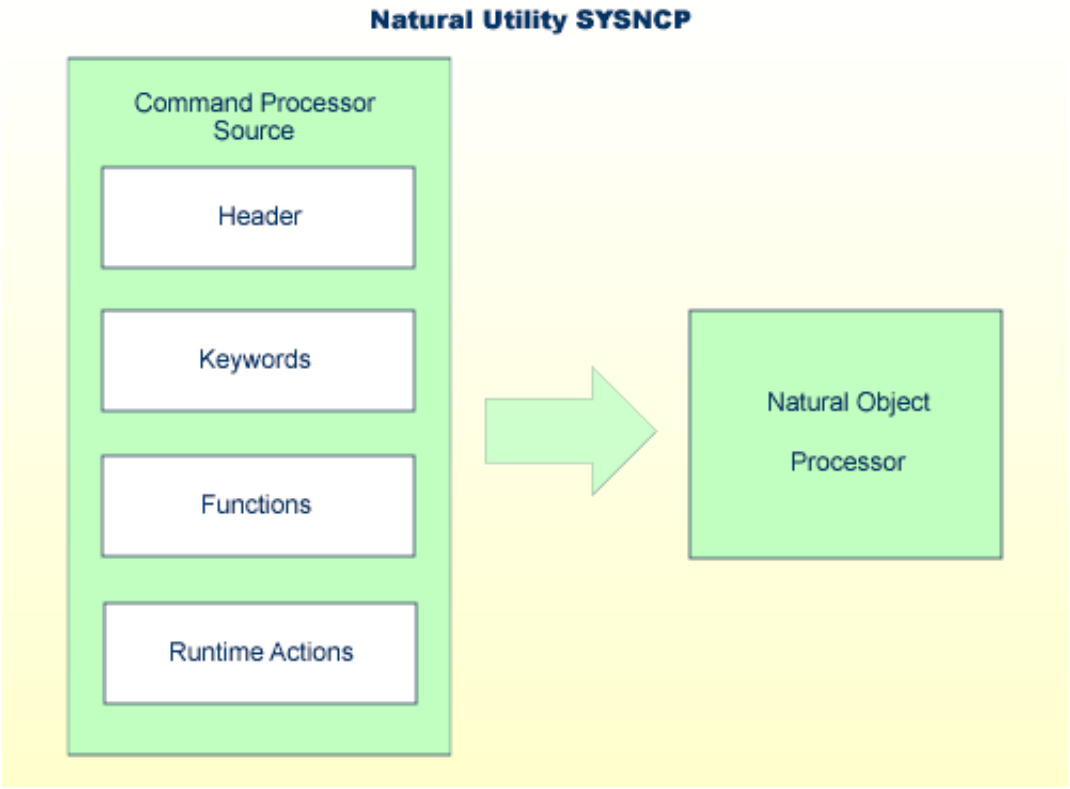


The location from which a certain command can be issued can be restricted on a system-wide or user-specific basis. On a system-wide basis, for example, the functions specified within commands can be local or global. A global function can be issued from **any** location while a local function can only be issued from specified locations. Restrictions can be placed on keywords and functions, however, if Natural Security is active in your environment.

Creating a Command Processor

The utility SYSNCP is used to create and maintain command processors. A command processor contains decision tables which determine what happens when a user enters a valid command.

The creation of a command processor is a cumulative operation involving several steps, from header definition, which establishes general defaults for the processor, to keyword definition, function definition and the linking of actions to functions. Special editors are provided by SYSNCP for the purpose of specifying keywords, functions and actions.



The end product of command processor development is a complex command processor source, which, when cataloged, generates a Natural object of type Processor. Whenever this object is referenced by the Natural statement PROCESS COMMAND, the runtime system of the Natural command processor is triggered.

Steps in Command Processor Creation

The following is a summary of the steps necessary to create a command processor.

To create a command processor

1. Verify/Modify the Session Profile.
   SYSNCP itself uses a Session Profile which contains various parameters which control how SYSNCP is to perform certain actions and how information is to be displayed. Desired modifications can be made and the resulting profile can be saved with a given user ID. See the section Session Profile.
2. Initialize the Command Processor.
   The name of the command processor and the library into which it is to be stored are specified.
3. Define Global Settings (Header).
   Various global settings for the command processor are defined. For example, descriptive text for keywords during editing, minimum and maximum length for keywords, in which sequence keywords are to be processed at runtime, runtime error-handling, and whether PF keys can be used at runtime to invoke functions. See the section Header Records.
4. Define Keywords.
   Each keyword which is to be processed by the command processor is defined together with an indication as to whether the keyword is to be entered as the first, second or third entry of a command.
   Keyword synonyms can also be defined as well as parameter indicators. User text can be defined for each keyword. This text can subsequently be read at runtime using the PROCESS COMMAND ACTION TEXT statement. See the section Keyword Maintenance.
5. Define Functions.
   Functions are defined by validating keyword combinations. A function can be defined as local (can only be invoked from a specific location within an application) and/or global (can be invoked from anywhere within an application). See the section Function Maintenance.
6. Define Runtime Actions.
   The actions to be taken by the command processor when a command is issued at runtime are specified. Example actions are: fetch a Natural program, place a command at the top of the Natural stack, place data at the top of the Natural stack, change contents of the command line. See the section Runtime Actions.
7. Catalog Command Processor.
   The resulting source is cataloged as a Natural object (type Processor) in the designated Natural library. The command processor can now be invoked by a Natural program using the PROCESS COMMAND statement. See the section Processor Cataloging.

